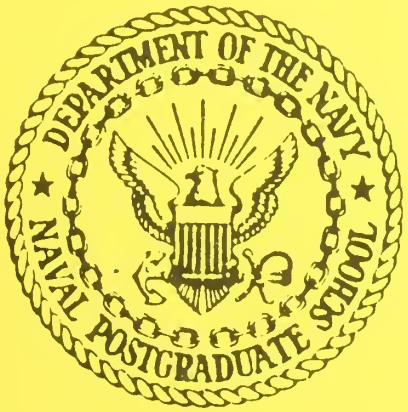
NPS52-86-013

NAVAL POSTGRADUATE SCHOOL

Monterey, California



A RESEARCH REPORT ON THE
LABORATORY FOR DATABASE SYSTEMS RESEARCH:
PAST, PRESENT AND FUTURE*

Jagdish Anand, Steven A. Demurjian, David K. Hsiao
Vincent Y. Lum, Dana E. Madison, Roger G. Marshall
and C. Thomas Wu

May 1986

Approved for public release; distribution unlimited

Prepared for:

Chief of Naval Research
Arlington, VA 22217

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral R. H. Shumaker
Superintendent

D. A. Schrady
Provost

The work reported herein was supported by Contract from the Office of Naval Research.

This report was prepared by:

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: NPS52-86-013
2. GOVT ACCESSION NO.:
3. RECIPIENT'S CATALOG NUMBER:

4. TITLE (and Subtitle): A Research Report on the Laboratory for Database Systems Research: Past, Present, and Future*

5. TYPE OF REPORT & PERIOD COVERED:

6. PERFORMING ORG. REPORT NUMBER:

7. AUTHOR(s): Jagdish Anand, Steven A. Demurjian, David K. Hsiao, Vincent Y. Lum, Dana E. Madison, Roger G. Marshall and C. Thomas Wu

8. CONTRACT OR GRANT NUMBER(s):

9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Postgraduate School, Monterey, CA 93943-5100

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: PE62760N, R0533, 85PR35141, N001485WR35084

11. CONTROLLING OFFICE NAME AND ADDRESS: Chief of Naval Research, Arlington, VA 22217

12. REPORT DATE: May 1986

13. NUMBER OF PAGES: 24

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office):

15. SECURITY CLASS. (of this report):

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this Report): Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report):

18. SUPPLEMENTARY NOTES:

19. KEY WORDS (Continue on reverse side if necessary and identify by block number):

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The Laboratory for Database Systems Research at the Naval Postgraduate School is devoted to research and experimentation on a wide-range of database topics. The major goal of the Laboratory is to maintain a constantly evolving environment that can be utilized by professors and students to conduct research. In this paper, we report on our past, present and future research efforts at the Laboratory for Database Systems Research. The past and present research that we report on is in four major areas, architectures for high-performance database computers, portable database systems, user interfaces for database

systems, and methodologies for database systems. The future research we report on is in three areas, real-time database computers, multi-model database systems and multi-medium database systems. In all of our discussions, our main focus is on providing an overview of why we do our research with the secondary and rather limited focus on the actual details of the research.

A RESEARCH REPORT ON THE LABORATORY FOR DATABASE SYSTEMS RESEARCH: PAST, PRESENT AND FUTURE*

Jagdish Anand, Steven A. Demurjian, David K. Hsiao, Vincent Y. Lum,
Dana E. Madison, Roger G. Marshall and C. Thomas Wu

Department of Computer Science
Naval Postgraduate School
Monterey, California 93943
U. S. A.

ABSTRACT

The Laboratory for Database Systems Research at the Naval Postgraduate School is devoted to research and experimentation on a wide-range of database topics. The major goal of the Laboratory is to maintain a constantly evolving environment that can be utilized by professors and students to conduct research. In this paper, we report on our past, present and future research efforts at the Laboratory for Database Systems Research. The past and present research that we report on is in four major areas, architectures for high-performance database computers, portable database systems, user interfaces for database systems, and methodologies for database systems. The future research we report on is in three areas, real-time database computers, multi-model database systems and multi-medium database systems. In all of our discussions, our main focus is on providing an overview of why we do our research with the secondary and rather limited focus on the actual details of the research.

* The work reported herein is supported by grants from the Department of Defense STARS Program and conducted at the Laboratory for Database Systems Research, Naval Postgraduate School, Monterey, CA 93943.

1. AN INTRODUCTION

The Laboratory for Database Systems Research at the Naval Postgraduate School is devoted to research and experimentation on a wide-range of database topics. Founded at the Ohio State University and moved to its current location in 1982, the Laboratory personnel consists of four professors, three Ph. D. students and fourteen Master's students. The major goal of the Laboratory is to maintain a constantly evolving environment that may be utilized by professors and students to conduct research. In meeting this goal, we make an extensive use of the Laboratory facilities at the Naval Postgraduate School. In addition, this environment utilizes some of the computing equipment of the Department of Computer Science. The entire computing environment is supported by a staff provided by the Department.

In this paper, we report on our past, present and future research efforts at the Laboratory for Database Systems Research. The past and present research that we report on is in four major areas, architectures for high-performance database computers, portable database systems, user interfaces for database systems, and methodologies for database computers. Briefly, let us outline each of these areas.

Our research in architectures for high-performance database computers is motivated by our disappointment with traditional approaches to database computer architectures. As we show in Section 2, we have found that there are some serious limitations in the performance of traditional approaches. Therefore, we have sought to develop novel architectures for database systems that stress high-performance as a primary design goal. In Section 2, we present our two novel approaches to architectures for high-performance database systems. The first approach is hardware-based, while the second approach is software-based. Both approaches achieve high-performance by the utilization of parallelism to increase the throughput.

Our research with portable database systems has arisen from the recognition that database systems of the present and future must be able to run on different hardware and operating system configurations. No longer is it sufficient for a database system to be supported on a single type of hardware using a specific series of operating systems. Advances in networking and hardware technology have created a computing environment that is wide-ranging and variable. Consistent database support in such an environment mandates a database system that is capable of functioning on different hardware and operating system configurations. In Section 3 we report how we have designed and developed a highly-portable database system.

Our research on user interfaces for database systems is motivated by the need to provide new functionality, in terms of user access methods, to database systems. In Section 4, we investigate the two distinct approaches in our user interface research, both predicated by a desire to overcome the traditional limitation of the single-data-model-and-single-model-based-data-language paradigm used by most database systems. In the first approach, we have developed many new one-dimensional (or textual) user interfaces, based on different data models and data languages, to provide the user with a wide-range of database access methods. In the second approach, we are developing a two-dimensional (or visual) user interface to provide the user with a graphical framework for database accesses.

In our research on methodologies for database computers, we have determined that there is a need to provide analytical and empirical methodologies that may be used to qualitatively and quantitatively evaluate our database computer research. In the course of all of our different research efforts, we have found that at different stages of the research process (i.e., design, testing, and prototyping) we need methodologies that facilitate design analysis, performance estimation, design verification and performance evaluation. In Section 5, we explore five different methodologies that are used for analytical and empirical evaluations. We present two different classes of methodologies. The first class is on innovative applications of classical methodologies. The second class is on the development of new methodologies and on the use of these methodologies.

Finally, to complement our presentation of past and present research in Sections 2, 3, 4 and 5, we introduce three new research areas in Section 6, namely, real-time database computers, multi-model database systems and multi-medium database systems. Our major focus in this section is to provide insight into our present thinking and efforts in each of these areas.

2. HIGH-PERFORMANCE DATABASE COMPUTERS

The general-purpose computers, known as the mainframes, are machines that support program executions and numeric computations. On the other hand, the database work is non-numeric. The database work is different from scientific computing, since typically a lot of data is moved in and out of the computer, and its computation is minimal. Whereas a general-purpose mainframe is concerned with the execution of a stored program at an exceedingly high speed, a special-purpose database computer may not have any stored program. Instead, it must provide high I/O bandwidth so that multiple data streams coming from its database stores can be processed by the database processors readily.

In the remainder of this section, we examine architectures for high-performance database computers. First, we present a taxonomy of database computers, in order to motivate the need for developing high-performance database computers and to contrast our own work with others. Given this taxonomy, the second part of this section provides a more detailed presentation of our architectures for high-performance database computers.

2.1. A Taxonomy of Database Computers and their Approaches

A taxonomy for the database computers may be helpful to the reader to place our work in proper perspective, and to understand our motivation to work in this area. It may also allow the reader to relate the conventional and traditional approaches to database management with the new and unconventional approaches to database management.

2.1.1. The Traditional Mainframe-Based Approach

Traditionally, the *mainframe-based* approach to database management and processing have dominated the field of the database management system (DBMS), where DBMS runs as an application of the operating system on the mainframe . In such an approach, the user writes the transactions in a data language and submits the transactions to the mainframe for execution.

The operating system of the mainframe first causes DBMS to be executed, and then passes the transactions to the running DBMS. We note that in this setting DBMS must share the use and control of the physical resources with all of the other (non-database) applications (such as compiler-language programs) of the mainframe computer.

By sharing the use and control of the physical resources such as data channels, disk controllers and disk drives, DBMS cannot support very large sizes of databases effectively and efficiently. The ineffectiveness is due to the limitation in storage capacity where the disk space is also being used for other applications. The inefficiency is due to the reliance on a general-purpose and all-embracing operating system to provide disk I/O operations and control. The mainframe-based DBMS, thus, is adequate only for small, simple and stable databases. It cannot, however, support very large databases adequately due to its inability (a) to accommodate very large sizes, rapid growth, and complex applications, (b) to deliver desired performance with or without hardware upgrades and (c) to provide a low cost of upgrade, high level of diverse applications, and low level of disruption during upgrades.

2.1.2. The Conventional Single-Backend Approach

To overcome the problems of performance degradation and resource sharing and control, the database-system software is off-loaded from the mainframe to a separate, dedicated computer with its own disk system, known as the *backend* of the mainframe. This conventional approach is characterized by the architectural configuration where DBMS is placed in a dedicated backend computer with its own operating system, disk controllers and disk drivers. As far as the mainframe is concerned the presence of the backend appears to it as a peripheral system much like the communications *frontend* which handles terminals and serves as the gate-way to a computer network. In comparison with other peripheral systems of the mainframe computers such as the disk system, the tape system and the unit-record devices, the database backend consists of considerable software, firmware and hardware. This is because, in addition to the DBMS software, there is the need of interfacing software for the backend and mainframe computers.

The conventional single-backend computers are good for small, simple and stable databases. They differ from the traditional mainframe-based DBMS in that they allow more cost-effective upgrade and little disruption to other non-database applications. They also provide better physical protection of the databases and more incentives for retaining the use of the mainframe computers (i.e., hosts) for a longer period of time. Nevertheless, the problems and issues that have confronted the mainframe computers on very large database sizes, growth, complexity, performance and cost have not been resolved; they are, instead, merely being deferred to the single-backend computers.

2.1.3. The Hardware Machine Approach - The Database Computer

The major issue of the traditional mainframe-based or the conventional single-backend approach is the lack of performance and capacity, as the database sizes increase significantly. The conventional hardware and software systems also provide no solution to this breakdown in

performance and capacity even if the database size is stable, but very large. However, an unconventional *hardware-machine* approach -- the database computer (DBC) -- may provide a solution. DBC, as our design of a database computer, provides a hardware solution to the single-backend database computer. In this approach, it has been hoped that by realizing all of the DBMS functions in the hardware, the performance of the database computer would vastly be improved over the software-oriented single-backend database system. Consequently, the hardware-oriented DBC could handle large databases effectively and efficiently.

2.1.4. The Parallel Architectural Approach - The Multi-Backend Database System

In addition to the hardware machine approach to the single-backend database management, the use of *multiple-backends* in a parallel fashion can also overcome the limitations and shortcomings of the mainframe-based and single-backend DBMS. In this approach, all backends work concurrently. The backends are conventional mini-(or micro-)computers that are identical. The disk controllers and drives are local to the respective backends. The database is centralized and evenly placed on the respective disks of the backends. Such a system, called the multi-backend database system (MBDS), can be made to grow by the addition of the same hardware to support performance gains and capacity growth, without the need of hardware replacement and upgrade. Nor does it require reprogramming.

2.2. New and Unconventional Approaches to High-Performance Database Systems

In the taxonomy, we have gained a perspective of the new database computers as either hardware single-backend solutions (e.g., DBC), or software multiple-backend solutions (e.g., MBDS). In the succeeding subsections, we present the work done in these perspectives. In the design of both DBC and MBDS, we have tried to overcome the performance limitations of either the traditional software mainframe-based or conventional single-backend approach to database management with large databases. Thus, in both DBC and MBDS, the high performance has been stressed as a primary design goal.

2.2.1. The Database Computer (DBC)

The conventional hardware and software systems could not overcome the performance limitations when working with large databases. Thus, the requirements of DBC have enforced the design of a special-purpose machine. Functionally, DBC looks into two primary requirements: handling of the database and handling of the database operations. The characterization of these two functions have been instrumental in determining the design goals of DBC.

Architecturally, Figure 1 is a complete diagram of the major DBC components. DBC acts as a backend machine to one or more general-purpose computers which are jointly referred to as hosts. Users' programs reside in the host and are executed by the host using DBC as one of its various resources. The host communicates with DBC by way of DBC commands, and DBC responds either by returning a group of records (i.e., the response set), or by indicating successful or unsuccessful execution of a command.

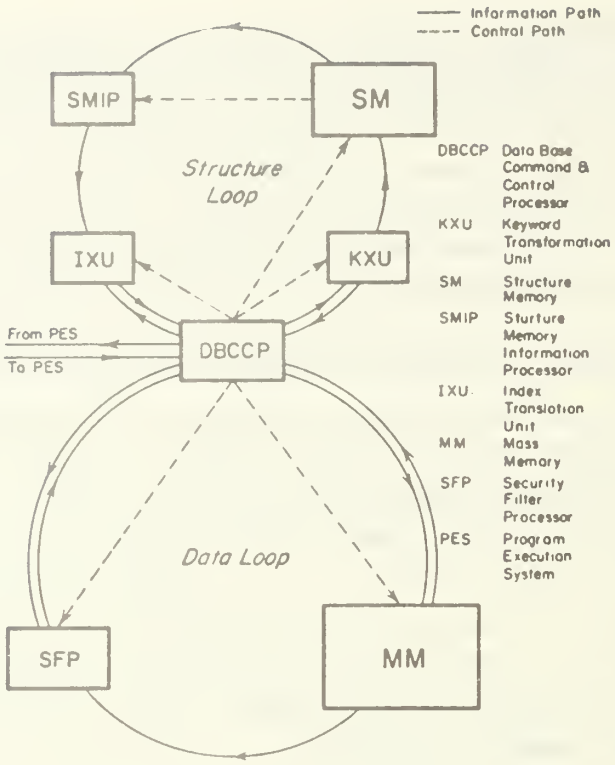


Figure 1. The Organization of DBC.

DBC makes use of two loops of processors and memories in executing the commands. The *data loop*, which consists of the database-command-and-control processor (DBCCP), the mass memory (MM), the post processor (PP), and the security-filter processor (SFP), is used for storing and accessing the database, for post-processing of retrieved records, and for enforcing field-level security. The *structure loop*, which consists of the database-command-and-control processor (DBCCP), the keyword-transformation unit (KXU), the structure memory (SM), the structure-memory-information processor (SMIP), and the index-translation unit (IXU), is used for limiting the mass-memory search space (through the determination of cylinder numbers), for determining the authorized records for accesses, and for clustering records received for insertion into the database.

Given our characterization of DBC, how can DBC achieve higher performance than either the traditional mainframe or conventional single-backend approaches to database management? To answer this question, let us consider the primary design goals of DBC, which are to achieve a high-degree of parallelism, concurrency and pipelining for performance gains. *Parallelism* is achieved at two levels in DBC. First, the indexing information for the database is evenly partitioned within the structure memory (SM), to provide index-serial-and-partition parallel access. Second, the database records are clustered (using the indexing information) and distributed evenly among the disk tracks of a cylinder in the mass memory (MM), to provide record-serial-and-track-parallel access. *Concurrency* is also achieved at two levels in DBC. First, at the transaction processing level, DBC is designed to allow the transactions of many users to be present in the system at one time. Second, DBC has the ability to overlap the index processing

of one transaction with the record processing of another transaction, resulting in an even lower-level granularity of concurrency. Finally, pipelining is used to complement parallelism and concurrency. *Pipelining* is attained concurrently in two parts of DBC. By having the index information that is needed by a transaction to flow systematically from one component to another component in the structure loop, i.e., from KXU to SM to SMIP to IXU, DBC pipelines the index processing. By having the records that are needed by the transaction to flow systematically from one component to another component in the data loop, i.e., from MM to PP to SFP to DBCCP, DBC also pipelines the record processing. The detailed design of DBC can be found in the references.

2.2.2. The Multi-Backend Database System (MBDS)

We have pioneered the *multiple-backend* approach to database management and processing with a parallel architecture of database processors and their database stores, i.e., backends. Unlike a fixed parallel system, where the number of processors once built-in cannot be changed, the processor-store pair of the multiple-backend database computer can be added and deleted without requiring any reprogramming of the system software or any modification of the system hardware. Thus, not only is this a *variably parallel* system, but the architecture naturally supports hardware upgrade with ease.

The design of MBDS, has been influenced by three primary goals, which are (a) *performance gains in terms of response time reductions*, (b) *capacity growth in terms of the response time invariance*, and (c) *system expandability*. In terms of the ease in adding the new hardware (i.e., backends) and in configuring the existing software, the third goal is met. By increasing the number of backends while the size of the database and the size of the responses to the transactions remain constant, MBDS produces a reciprocal reduction in the response times of the user transactions. By increasing the number of backends proportionally to the increase of transaction responses, MBDS produces invariant response times for user transactions. The first goal allows the multiplicity of the backends of MBDS to be directly related to the performance gains of MBDS in terms of the response-time reduction. The second goal enables the multiplicity of the backends of MBDS to be directly related to the capacity growth of MBDS in terms of response-time invariance.

Figure 2 illustrates the architecture of MBDS. When a transaction is received from a host computer or from a terminal, the controller broadcasts the transaction to all the backends. Each backend has a number of dedicated disk drives. Since the data is distributed on the backends' disks evenly, a transaction can be executed by all backends simultaneously and access the database parallely. Each backend maintains a queue of transactions and schedules a transaction for execution independent of the other backends, in order to maximize its access operations and to minimize its idle time. Thus, transactions are also executed in the backends concurrently. Nevertheless, the controller does very little work. It is responsible for broadcasting, routing, and assisting in the insertion of new data. The backends do all of the primary database operations.

Being a message-oriented system, MBDS is organized into a number of processes, both in the controller and in the backends. The controller processes are the communications processes,

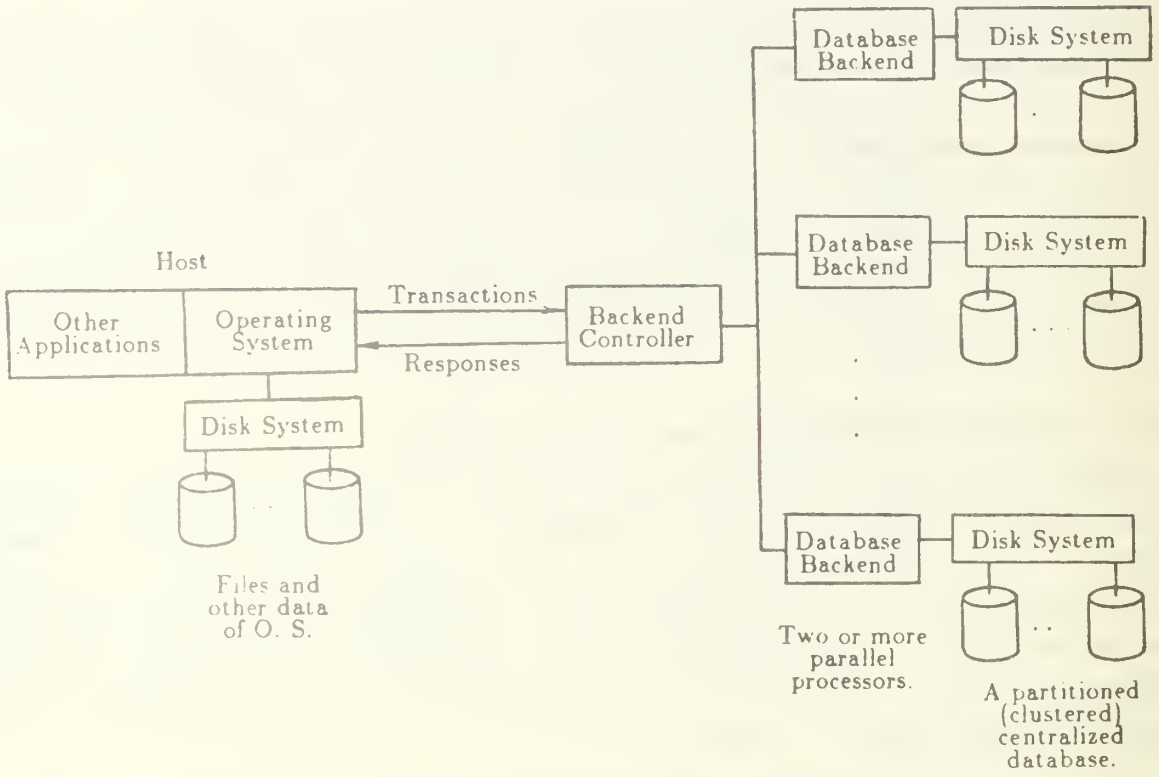


Figure 2. The Multi-Backend Database System.

the interface process, the request preparation process, the insert information generation process, and the post processing process. The communications processes, present in all computers, enables a message to be placed on (or to be received from) the communication bus. The interface process allows the user to interact and access the database system directly. The request preparation process receives, parses and formats a request (transaction). The insert information generation process arbitrates the record insertion process. Finally, the post processing process is used to collect all the results of a request (transaction). In addition to the communications processes, the backend processes are directory management, concurrency control, and record processing. Directory management performs the search of the directory tables to determine the secondary-storage addresses necessary to access the clustered records. Concurrency control is used to arbitrate the access of the directory data and user data. Record processing performs the disk I/O operations and other record operations specified by the request.

To achieve performance gains, the backend controller of MBDS does minimal work. The communications bus is of the broadcast type for ease of communications between the controller and backends. They are also cost-effective. Further, the communications between the controller and its backends and among the backends are minimal. To further improve the performance, the database is placed on the disks of the backends in such a way to facilitate subsequent access operations in the block-parallel-and-record-serial mode. Also, the directory is placed in the system in such a way to facilitate either parallel processing of a transaction or concurrent processing of several transactions or both. For detailed design and implementation of the prototyped MBDS, the reader may refer to the references.

3. PORTABLE DATABASE SYSTEMS

As database systems move into the future, they must be able to adapt to many different computing environments. Advances in technology have given rise to computing environments with the wide-ranging and various types of operating systems and computer hardware. In fact, in some cases, for the same hardware there may be two or more markedly different operating systems available. For a database system to play a major role in such an environment, the database system must have the ability to function on the different computers with different operating systems.

To achieve consistent database support in such a volatile environment, we must strive to develop a database system that is highly portable. What is a highly portable database system? A database system is *highly portable* if we can easily transport the database system software from one computing environment (characterized by a specific hardware/operating-system configuration) to another computing environment (also characterized by another specific hardware/operating-system configuration). In this transportation of database system software, we have a change of either the hardware or the operating system, or both. To attain a portable database system, we must strive to develop the database system software so that it contains a high degree of hardware and operating-system independence. By engineering the database system with a minimum amount of dependencies, we greatly enhance the *probability* that the database system software is easily transportable to new hardware/operating-system configurations.

In the remainder this section we show how we can design and develop a highly portable database system. First, we present the design issues which play a significant role in the implementation of a highly portable database system. Then, we present the three different hardware/operating-system configurations under which we have implemented our highly portable database system (i.e., MBDS).

3.1. A Highly Portable Database System

To develop a highly portable database system, we first need to identify which portions of the database system software are dependent on either the hardware and/or the operating system. There are two classes of database system software which are dependent, namely, communications software and disk input/output software. Communications software is used by the database system to communicate between different computers and to communicate within a computer, referred to as *inter-computer* and *intra-computer communications*, respectively. The communications software is often affected by a change of the operating system (since communications protocols are operating-system dependent) and is also affected by a change of the hardware (since specialized communications drivers are hardware dependent). The *disk input/output* software is used by the database system to access and process information from the secondary storage. The disk input/output software is also affected by a change of the operating system (since it is operating-system dependent) or by a change of the hardware (since it is dependent on specialized disk drivers).

In general, there is no way that we can avoid a certain amount of hardware and operating-system dependencies in a database system. Instead, we should develop techniques which can

minimize the effect of changes. There are two distinct approaches to accomplishing this task. First, we use the concepts of *abstraction* and *encapsulation* to isolate the dependencies of the communications and disk input/output software. The database system software makes calls to these high-level routines that are dependent on the programming language used in the software system, when we need to access the system-dependent software. These calls are generic, e.g., send[message, destination], receive[message, sender], do_disk_io[data, device]. They represent abstractions of the actual functions. The routines themselves (i.e., send, receive, and do_disk_io) are used to encapsulate the system-dependent software. Second, we use the concept of a *virtual interface* to develop independent software for communications and disk input/output. The aim of the virtual interface is to utilize abstractions provided by the compiler to accomplish a particular task. These abstractions are usually in the form of library routines for the programming language. As these library routines are supported by different compilers under different operating systems, we can easily transport the virtual interface from one operating system to another.

In the multi-backend database system (MBDS), we have utilized the abstraction and encapsulation, as well as the creation of a virtual interface. Abstractions and encapsulations are used by our communications software to provide high-level calls to send and receive messages both between and within computers. Since MBDS is a message-oriented system, all of the inter-computer and the intra-computer communications are accommodated by the abstractions and encapsulations. These techniques are also used by our disk input/output software to provide a high-level interface for reading (writing) information from (to) the secondary storage. In addition, we have also created a virtual interface for our disk input/output software. The virtual interface depends on the programming language constructs (in this case, the C language constructs), and is used to provide a high-level, operating-system-independent paradigm for performing disk input/output via text files.

3.2. The Current Configurations of the Highly Portable Database System

In our research, we have implemented three different hardware/operating system configurations for our highly portable database system MBDS. The first configuration is the original configuration. The controller for MBDS is a Digital Equipment Corporation (DEC) VAX-11/780 (with the VMS 3.7 operating system). The backends for MBDS are two DEC PDP-11/44s (with the RSX-11/M operating system). For the database store, each backend utilizes a 67-megabyte RM03 disk drive. Communications is accomplished using the point-to-point parallel communications link (PCL), a 0.5-megabit bus. Three PCLs are utilized, one from the controller to the first backend, one from the controller to the second backend and one between the two backends. The inter-computer communications software of both the controller and the backends uses the DEC PCL driver. The intra-computer communications software in the controller uses VMS mailboxes, while shared memory is used in the backends, i.e., the PDP-11/44s. The disk input/output software in the backends use RSX provided low-level input/output routines.

For the second configuration, we have changed both the hardware and the operating system, resulting in the most drastic effect on porting. In the second configuration the controller for MBDS is a DEC VAX-11/750 (with the 4.2 B.S.D. Unix operating system). The backends for MBDS are seven Integrated Solutions Incorporated (ISI) Motorola 68020-based workstations (also with the 4.2 B.S.D. Unix operating system). For the database store, each backend utilizes a 500-megabyte Control Data Corporation (CDC) winchester-type disk drive. Communications is accomplished using an Ethernet. All computers (both controller and backends) share the same Ethernet. In this porting, we have encapsulated and modified the inter-computer communications software (to use Unix TCP/IP) of both the controller and the backends, the intra-computer communications software of both the controller and the backends (to use Unix sockets), and the disk input/output software (to do raw I/O to a Unix file system). The implementation of the second configuration occurred in a number of stages or versions. During one of the stages, we have developed a virtual disk input/output abstraction to replace the system dependent software used in the first configuration. This abstraction depends only on the C compiler, i.e., is independent of the Unix operating system.

With the experience gained from the first and second configurations porting, we proceed with the third configuration. In the third configuration, the controller for MBDS remains a DEC VAX-11/780 (but with the 4.2 VMS operating system) while the backends are upgraded to DEC MicroVax-IIs (with the 4.1 MVMS operating system). The communications bus is also upgraded to a DELNI, with DECNET providing the networking software interface. Each backend has a 71-megabyte DEC winchester-type disk drive. In this porting, we modified the inter-computer communications software (to use DECNET protocols) of both the controller and the backends, the intra-computer communications software of the backends (to use the mailbox facility provided in VMS) and the disk input/output software. Since disk drivers are operating system dependent in the Microvaxes, we use the virtual disk input/output abstraction of the second configuration for our disk input/output software of this configuration.

4. USER INTERFACES FOR DATABASE SYSTEMS

Unlike physical resources of a computer where one piece of a physical resource (say, a reel of blank tape) does not have information related to another piece of the physical resource (say, another reel of blank tape), data in a database are used to represent related information. In fact, most of these relationships are also represented in data. This is because we are not merely using the data as information items (i.e., physical resources),but we are also using the data for related information (i.e., logical resources) so that we can process the related data for our transactions and manipulate the relationships for deriving new information and relationships.

Data models have been used to represent the relationships. In a contemporary database management system (DBMS), where the database is small and simple, DBMS supports only a single data model and model-based data language. Consequently, we have, for example, four separate DBMSs, one is the relational DBMS which supports the relational data model and relational data language, one is the hierarchical DBMS which supports the hierarchical data model and hierarchical data language. one is the CODASYL DBMS which supports the

CODASYL data model and CODASYL data language and one is the functional DBMS which supports the functional data model and the functional data language. For different applications, we may thus use, for example, a relational DBMS for handling tables, forms, and ad hoc queries, an hierarchical DBMS for managing designs of assemblies, subassemblies, components and parts, and a CODASYL DBMS for exercising inventory control of supplies and demands, and a functional DBMS for integrated design and manufacturing processes.

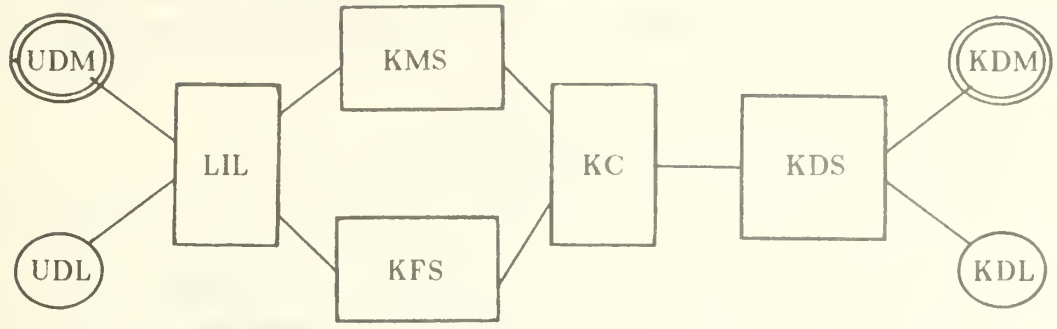
To meet the database system requirements of today, where the database applications are diverse, and involved, there may be applications, for example, in table handling, design management, inventory control and process integration. A single data model and model-based data language will not suffice, since the model and language that is good for one application may not be adequate for another application. Thus, what we provide in our system is the flexibility to support a variety of data models and a large number of model-based data languages. With the ability to handle multiple models and languages, the system allows the user to explore the strong points of these models and languages for their applications. Consequently, stored data and written transactions may best be developed for the intended applications. We call this approach, the multi-lingual database system (MLDS), i.e., *one dimensional* (or textual) capabilities of a database system, to allow the database system to support many-data models and their model-based data languages.

On the other hand, we are also interested in introducing new types of interfaces to database systems, to augment the one-dimensional approach. Referred to as *two-dimensional* (or graphical) interfaces, these interfaces allow the user to design new data models, languages and applications, in a pictorial framework. We call this approach, the graphics language for database systems (GLAD). Both of these approaches are discussed in the following sections.

4.1. The Multi-Lingual Database System (MLDS)

The system structure of a multi-lingual database system is shown in Figure 3. Users issue transactions through the language interface layer (LIL) using a user-chosen data model (UDM) and written in a corresponding model-based data language (UDL). LIL then routes the user transactions to the kernel mapping system (KMS). KMS has two tasks. First, if the user specifies that a new database is to be created, KMS transforms the UDM-database definition to an equivalent kernel-data-model-(KDM)-database definition. The KDM-database definition is then sent to the kernel controller (KC). KC sends the KDM-database definition to the kernel database system (KDS). Upon completion, KDS notifies KC, which in turn, notifies the user that the database definition has been processed and that the loading of the database may commence.

The second task of KMS is to handle UDL transactions. In this situation, KMS translates the UDL transaction to an equivalent kernel-data-language (KDL) transaction. KMS then sends the KDL transaction to KC, which in turn, sends the KDL transaction to KDS for execution. Upon completion, KDS sends the results in KDM form back to KC. KC forwards these results to the kernel formatting system (KFS) for transforming them from the KDM form to the UDM form. After the data is transformed, KFS returns the results, i.e., the response set, to the user via LIL. There is one final note of importance on the general system structure. Four of the five



UDM : User Data Model
UDL : User Data Language
LIL : Language Interface Layer
KMS : Kernel Mapping System
KC : Kernel Controller
KFS : Kernel Formatting System
KDM : Kernel Data Model
KDL : Kernel Data Language
KDS : Kernel Database System

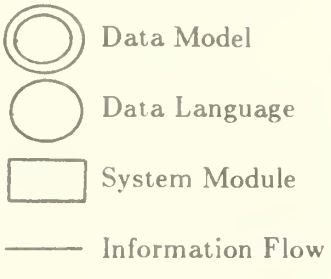


Figure 3. The Multi-Lingual Database System (MLDS)

components of the multi-lingual database system. namely, LIL. KMS, KC. and KFS, are referred to as a *language interface*. A new language interface is required for each chosen data language.

In our work on MLDS. we have developed four language interfaces, (i.e., the SQL, DL/I, CODASYL-DML and Daplex language interfaces) implemented on a VAX-11/780 running the 4.2 B.S.D Unix operating system. All of the modules of each language interface have been coded using the C programming language. The size of each language interface ranges from 3,000 to 4,000 lines of code. Initially, the interaction between the language interface and the kernel database system (KDS) is simulated. After each language interface is thoroughly debugged and tested, it is then integrated with KDS, which is, in fact, MBDS. The integrated version of three language interfaces is currently operational at the Laboratory for Database Systems Research. The Daplex language interface has not been completed as of this writing and therefore is not being integrated at this time. We expect to complete it within six months.

What is the importance or relevance of our work on MLDS? The issues and merits of MLDS fall into three categories, practical merits, new functionalities and theoretical issues. The major practical merit of MLDS involves the reusability of database transactions developed on existing database systems. The ability to reuse existing transactions reduces the recoding and redevelopment of database transactions in the new MLDS environment. The crucial new functionality of MLDS is to allow the new users to explore the strong points of different data models and to utilize desirable features of different data languages for their applications. With this capability, a wide and varying range of applications can be supported in one environment.

Finally, there are theoretical issues that may be studied, namely, the data-transformation process from UDM to KDM and the data-language translation process form UDL to KDL. By studying these transformations and translations, we can gain insight into the relationships between data models and data languages.

4.2. The Graphics Language for Database System (GLAD)

The system structure of *G*raphics *LA*nguage for *D*atabase System is shown in Figure 4. The database schema similar to the one shown in Figure 5 will be displayed on the screen, and users directly manipulate this database schema for querying the database. Users also interact graphically with the system to define the database schema. The data-definition-and-manipulation interactions of users are interpreted by the command interpreter. Interpreted commands are then passed to the query/data definition processor for the actual execution of commands. The result is passed to the display formatter or the internal data manager or both. The display formatter responds to the user command by changing the screen display. The internal data manager makes the actual changes in the database and/or data dictionary. This internal data manager may be a tailored-made system, off-the-shelf DBMS, or MBDS. Notice that the display formatter and command interpreter are independent of the model for an internal data manager.

The formal specification and partial implementation of the display formatter and the initial design of the command interpreter have been done by the Master's students. Our current plan is to fully implement the display formatter first and then develop the command interpreter and query/data definition processor simultaneously.

A goal of this research is to develop a coherent interaction method for all three user interactions with a database: the data definition interaction, the data manipulation interaction, and the program development interaction. Such user interface is critical in expanding the application areas of database management system technology. When the advanced application areas such as office automation and CAD/CAM/CIM are considered, it is apparent that a good user interface plays a major role in making DBMS acceptable to the users in these application areas. When the application becomes more complex, it is a must to have a user interface that is easy to learn and use. No matter how fast the system performs, if it is difficult to use, then the users will not adopt the system.

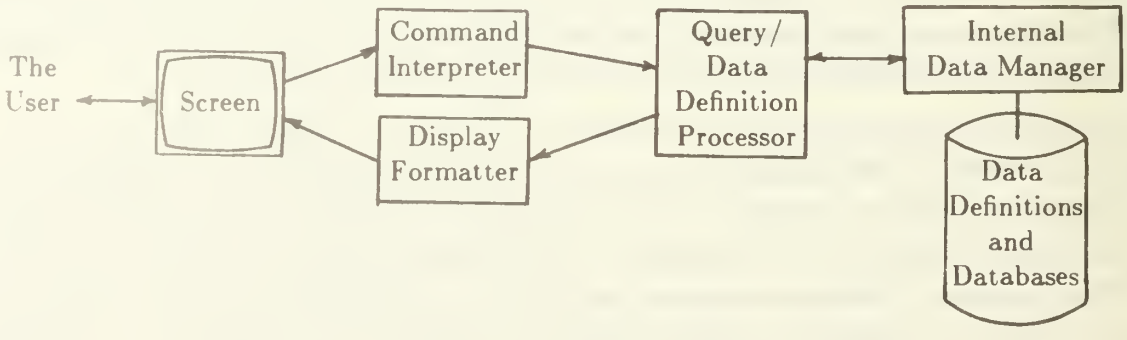


Figure 4. The Graphics Language for Database System.

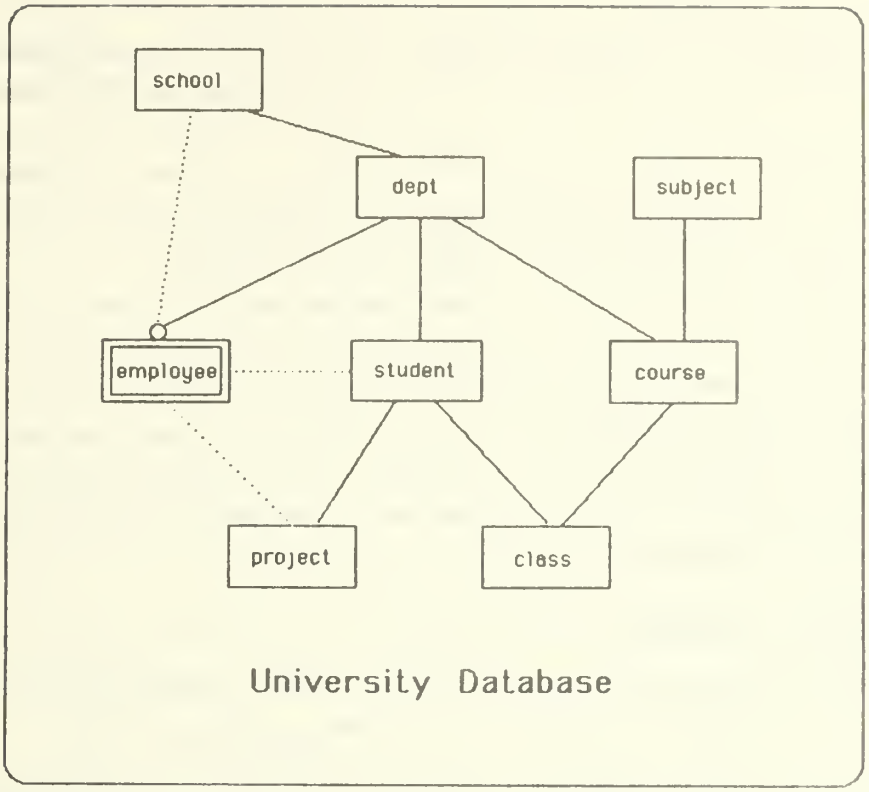


Figure 5. The Database Schema as Seen by the User.

5. METHODOLOGIES FOR DATABASE COMPUTERS

The development of suitable methodologies to facilitate design analysis, performance estimation, design verification and performance evaluation is a critical part of any research activity at every stage in the life-cycle development of a system, i.e.. requirements definition, specification, design, implementation, testing. How are these techniques developed and used in the course of designing and prototyping a database computer? Initially, there is a need to analyze the database computer at the requirements definition phase and continue it through the specification phases, i.e., the *design analysis.* Next, there is a need to estimate analytically the performance of the database computer, considering the various design alternatives available to decide on a particular design for implementation i.e., the *performance estimation.* Then, in the testing phase, there is a need to verify that the database computer indeed does what it is supposed to do, i.e., the *design verification.* Finally, there is a need to evaluate the absolute and relative performance of the database computer. once a prototype has been built, i.e., the *performance evaluation.*

Having established the need for methodologies for design analysis. performance estimation, design verification and performance evaluation, the next logical step is to identify the methodologies. The methodologies may be broadly classed into two classes. The first class comprises the mathematical and analytical methods to estimate the database computer performance and analyze design alternatives. They include queueing network models, time complexity analysis, and simulations. *Queueing network models* are a design analysis methodology that are used mainly at the requirements definition and specification phases of the

database computer development. The *time complexity analysis* is both a design analysis and a performance estimation methodology, whose usage overlaps in the specification and design phases. *Simulations* are used as a performance estimation and design verification methodology. They are used during the design and early implementation phases of the life cycle. The second class of methodologies comprises the empirical methods to evaluate the performance of the database computer. They include *benchmarking* and *checkpointing*. Both of these methodologies are quantitative, and are used after the prototype has been developed.

Our utilization and application of the methodologies falls into two broad categories. In the first category, we have innovatively applied existing methodologies (i.e., queueing network theory, time complexity analysis and simulations) for the design analysis and verification of both DBC and MBDS. In the second category, we have developed completely new methodologies (i.e., benchmarking and checkpointing) for the performance evaluation of database computers, namely, MBDS. In the remainder of this section we show how we have used all five of these methodologies in our work on DBC and MBDS.

5.1. Queuing Network Models

Queueing network models have been used extensively in modeling computer systems for design analysis and performance estimation. In the design analysis of DBC, we have used a *G/G/1 open network queuing model* on a specialized hardware component, that can perform efficient relational joins. The application of the model has helped in arriving at certain closed-form equations that can be used to determine and estimate the various design and performance parameters, i.e., the speed of the processors and the size of the memories, for the hardware component. For MBDS, eight directory management strategies for indices and all phases of the transaction execution have been analyzed using a *M/G/1 closed queuing network model*. The application of queueing models has been instrumental in determining the "best" directory management strategy and the "best" transaction execution strategy, which are, in fact, currently being used in the prototyped MBDS.

5.2. Time Complexity Analysis

Time-complexity equations may be used for the *case study* of an algorithm, e.g., the *best-case, average-case and worst-case times* for the algorithm. or they may be used to provide a *relative analysis* of the time spent in different implementations of the same algorithm. Time complexity analysis, in case of DBC. has been done on the specialized DBC component for the join operation. The analysis has been done by first examining the complexity of the DBC join operation, in terms of its execution sequence. Then, using the same techniques we have compared to and contrasted with the different time complexities of the join operations of other database machines. Our analysis has shown that the performance of the join algorithms for different database machines is comparable to the performance of the DBC join algorithm. In the case of MBDS, time complexity analysis has been done on the eight directory management strategies by developing case studies, i.e., the average-case-time and worst-case-time equations for each of the eight directory management strategies and then substituting typical values for the test variables to determine which of the eight strategies is the best. In this way, we have augmented our

queueing network model analysis with a relative analysis of the different implementation strategies of the directory management.

5.3. Simulations

Simulations, as a methodology, relieve some of the skill requirements for modeling the database computer. In building a *simulation model,* we are aided by the *simulation language*. We can simulate the database computer at any level of detail or abstraction. Consequently, for a database computer, it is easier to build a simulation model than to construct a queueing model. However, the input parameters of a simulation model are largely dependent on the experimenter's expert and judicious selections. In order to make any simulation study a useful and relevant endeavor, the objectives and goals of the study must be clearly specified. In our case, simulation techniques have been used to verify our intuitive notions on the performance of DBC, and help provide a framework for the development of MBDS. The main benefits of these simulation studies have been to answer some major design questions on the overall performance of the DBC, as well as the individual performance of the components of the DBC. In particular, we used simulations to verify the design goals of DBC (see section 2.2.1 again) and of MBDS (see section 2.2.2 again).

5.4. Benchmarking

Benchmarking involves the design and generation of test transactions and test databases for prototyped database computers. The tests must be thorough and be able to establish *standards* (i.e., *benchmarks*) for the performance or throughput of the database computer. Our first benchmarking methodology is developed to evaluate the performance of any relational database computer. Our second methodology is developed to evaluate a more specific class of computers, namely, multiple-backend database computers. In both methodologies, the key concern in the benchmarking of a computer is the specification of the workload. The *workload* of a database computer is characterized by three models that are hierarchically dependent: a model of the machine, a model of the database and a model of the applications. Therefore, to adequately develop a fair and unbiased benchmark set, the workload must be machine-independent. database-independent. and application-independent. To achieve machine-independence, the benchmark is constructed without bias toward any particular hardware organization or software architecture. To achieve database-independence, the benchmark database is independent of the database model of the real-world database. And, to achieve application-independence, the benchmark applications are generic.

We have used our first methodology for benchmarking the Britton-Lee database machine and the second methodology for the multiple-backend database computer MBDS. The benchmark database in MBDS is independent of the MBDS data model, and evenly distributed amongst all the backends. The benchmarks used to collect the timings are transactions which utilize the most prevailing and primary operations of the database system. The basic measurement statistics used in the benchmarking of MBDS is the response time of the transaction that is processed by the database system. Our preliminary benchmarking results indicate that MBDS is able to relate the multiplicity of the backends directly to the performance

gains and capacity growth of the system. More specifically, the response-time reduction for performance gain and the response-time invariance during capacity growth can be realized with the expansion of the system through the use of duplicated backend hardware and replicated backend software. Also, a set of tools are available to aid in the development of a test database. These tools include a test-file generation package and a database load subsystem. And finally, a set of tools that consists of the benchmark-generation package is available. The benchmark-generation package is used to create and execute benchmarks, and provides for easy variance in the types and complexity of benchmarks. This package also archives the benchmarks for later use.

5.5. Checkpointing

We refer to the benchmarking of a database system that collects only the response-time of the benchmarks as *external performance measurement*, since the database system is evaluated from a macroscopic view. However, when benchmarking a database system we are not only interested in its black-box performance, but we are also interested in the individual performance of the various database system components. *Internal performance measurement* is used to test the system from a microscopic view, in order to identify the distribution of work within the database system, which may in turn be used to fine-tune the overall performance of the database system. For example, internal performance times could include the time spent to search the indexes of the database, the time spent to stage the records from the database store into the primary memory, etc. Collectively, external and internal performance measurements of database systems are the basis of the development of a new *checkpointing methodology* which are the necessary tools for conducting benchmarking.

We have applied the checkpointing methodology to MBDS. MBDS has been instrumented with additional software tools that allow the enabling/disabling of the checkpoints. Second, data collection and statistical calculation tools have been provided. The collected data is directable to either the terminal or a file. In the instrumentation of MBDS for checkpointing we have tried to ensure that, (1) additional checkpointing software is incorporated into MBDS without any ill effect on the MBDS performance, (2) the checkpointing software can easily be toggled to allow the running of MBDS unimpeded, (3) the overhead of the checkpointing software is made available and (4) both raw and aggregated timing data are made available, via the software.

A monograph on these five methodologies and their application is being published by Prentice-Hall this year. The reader may refer to it in the references.

6. FUTURE DATABASE COMPUTERS

Finally, we would like to note our future research areas. We are currently investigating three major areas at the Laboratory for Database Systems Research, namely, real-time database computers, multi-model database systems and multi-medium database systems. By providing an overview of our present thinking and efforts in each of these areas, we hope to provide the reader with a final look at our research process.

6.1. Real-Time Database Systems

The real-time computers have applications in surface-to-air, air-to-surface, surface-to-surface and air-to-air weapon systems. However, very little basic research has been conducted on the fundamental and generic issues of the real-time computers. What have been practiced are the design, implementation and fine-tuning of specific real-time computers for certain applications. These pursuits, although necessary, do not provide fundamental understanding of the issues on hand. Nor do they provide fundamental contributions to the advancement of real-time computers at large. In our proposed research in this direction, we are looking forward to addressing generic issues and making fundamental contributions in real-time computers. In this study, we intend to focus on the *real-time database computers* which are characterized by the presence of large amounts of data and a number of databases. Furthermore, the computers are driven by the input data and stored data to meet the time constraints of a combat situation. Our focus, therefore, shall be on the design, analysis and prototyping of real-time database computers, to be used as backend computers that deal with three sets of data, i.e., three types of databases:

(1) the well-identified, clearly-classified, properly-screened and highly-resolved data, i.e., input (target) data,

(2) the reference data consisting of locations, positions, types and other data (not target data) of interest, and

(3) the operational data for the computation and generation of responses (known as the firing control data).

The approach to the development of a real-time database computer will mimic the development efforts of DBC and MBDS outlined earlier in this paper. In the course of the development, there will be the need of a *temporal data model*. This may include the specification of the new data model for real-time data and the application of existing and the development of new methodologies to evaluate our modeled computer.

6.2. Multi-Lingual-and-Multi-Model Database Systems

In the current version of MLDS a user of the relational/SQL language interface creates a database which is only accessible via SQL or KDL. Similarly, hierarchical databases are only accessible via DL/I or KDL, network databases are accessible via CODASYL-DML or KDL and functional databases are accessible via Daplex or KDL. However, we believe that there is a need to have a system that would allow one-model-based transactions to access other-model-based databases. For instance, can we allow the user of the Daplex language interface to access a CODASYL database? The implications of such a system are profound.

6.2.1. Towards Ultimate Database Sharing

By allowing the databases based on different data models to be accessed by data languages based on different data models, we extend our multi-lingual database system to a multi-model database system as well. Our present work indicates that a truly multi-lingual-and-multi-model database system (MLMMDS) is in sight. Such a system will allow ultimate sharing of databases

created via different models and languages. Our current work in this area is in two distinct, yet related, directions. First, we are pursuing the avenue of allowing the CODASYL-DML user to access a functional database created by Daplex. Second, we are examining the ability of the multi-lingual-and-multi-model database system to provide information support for the integrated design and manufacturing process.

6.2.2. Towards Rapid Information-System Support

There are some interesting reasons why we feel that MLMMDS may be used in information support for the integrated design and manufacturing process. The standard approaches to determining the information support occurs in three steps; (1) develop a model which characterizes the integrate design and manufacturing process, (2) specify an implement a model-based language for the definition, operation and control processes, and (3) instrument an *information system* in support of the model and language for these processes. However, the standard approach has a number of shortcomings as outline below:

(1) The life cycle of model development, language specification and implementation, and information-system instrumentation is long, i.e.. time consuming.

(2) There are high risks that either the model may turn out to be inadequate, the language ineffective and the system inefficient. There are too many compounded errors and cascaded mistakes.

(3) Repeating the life cycle with a new model, a new language and a new system may be time-consuming also.

MLMMDS is an alternative to the standard approach. and offers the following advantages:

(1) To try the existing data models and languages in the multi-model and multi-lingual database system until the designer can determine the adequacies and inadequacies of the existing data models and languages in providing the necessary design support for the intended system.

(2) To select or modify an existing data model and language in order to tailor them for the intended system, if the data model and language have many merits and only a few shortcomings.

(3) To specify and design a new data model and language with the experience gained in experimenting various existing models and languages for the intended system.

6.3. Multi-Medium Database Systems

In the past, DBMS applications have been limited to the business or commercial environment. As a result, nearly all the DBMSs have been developed with this environment in mind. Their success can indeed be claimed in this environment. However, because the systems have been targeted explicitly for this segment of applications, they lack the characteristics needed elsewhere. For example, commercial DBMS's support normally only formatted data with very few data types; they support only short fields; they match the whole content of a field to retrieve data; etc. These are too restrictive for other applications.

As the database field becomes more and more mature, applications become more and more diversified. For example, not only may the user want to store the formatted data in a DBMS, but

he may want to store other kinds of data, such as text, graphics, images, signals. and voices, as well. These data differ greatly in many aspect from the formatted data that we know how to handle well. These kinds of data are handle very poorly by a typical DBMS. To understand a little more what is meant, let us discuss a little more in some detail.

First, let us take the text data. The text data is the easiest to handle in the newer kinds of data. This is because the text data is much more structured than the others. Yet, even a simple extension to handle the text data differently from the formatted data causes many problems. First, as opposed to short field lengths in the formatted data, the text data tends to be long. The conventional DBMS cannot handle long fields. But more importantly, the contents of a text need to be searched, not by matching the whole field as we do for the short, formatted data fields, but by looking for only a short segment of the content. While indexing techniques developed by researchers in the library sciences may sometimes be applied, these techniques do not work well with the DBMS environment. For example, in a library environment, long searches are acceptable, updates are not on-line, inaccurate answers are useful, etc. This sort of thing is not acceptable at all in the normal DBMS applications. New techniques need to be developed to solve many such problems. While some of the proposals such as those mentioned in the references are addressing some of the problems, a broadly applicable solution is still not yet in sight.

Problems of other kinds of data are much more severe. In graphic data, one has hope of classifying the data using geometric patterns. Here, we make a distinction between the graphic data and the image data in that the graphic data is generated by some artificial way as in engineering drawings and the image data resembles the natural form of a tree or landscape. Even with this distinction, to have the computer to handle the graphic data is an order of magnitude harder than handling the text data. Here, even the problem of handling long fields can be difficult as these fields can also be very large and the performance of a DBMS is hardly satisfactory. In fact, in all of the CAD/CAM systems in production environments, their data is usually handled by a file system built specially for that system. Even in cases where a DBMS is used, the DBMS is merely serving as a file server, and not much more.

For the image data, the performance problem is much more severe as each image can take millions of bytes to represent it. Further, the characteristics of images are much harder to classify. For example, how are we to define the characteristics of a tree? Thus, to solve the classification problem, we need to look for AI techniques in order to help us.

As we proceed to deal with the voice data and the signal data, the problems are increasingly harder to solve. Here, even similarities among objects may not mean that we are dealing with similar objects. At the same time, the dissimilar data may mean the same objects. Such case occurs, for example, when one records the patterns of a phrase uttered by persons. As people do have different accents and voices, similar patterns may mean different phrases and dissimilar patterns may mean the same. In fact, today there is little research being done in this area of database applications.

In short, the diversified DBMS applications have created many problems and therefore, research opportunities. In order to allow these diversified applications, it is necessary for us to solve these problems first. Solutions to these problems probably will need technologies from

different disciplines and is expected to require both hardware and software solutions.

SELECTED REFERENCES

On the database computer, DBC:

Banerjee, J., Buam, R. I. and Hsiao, D. K. (1978). Concepts and capabilities of a database computer. *ACM Transactions on Database Systems,* Vol. 3, No. 4.

Banerjee, J., Hsiao, D. K. and Kannan, K. (1979). DBC - a database computer for very large databases. *IEEE Transactions on Computers*, Vol. C-28, No. 6.

On the database computer, MBDS:

Boyne, R. D., Hsiao, D. K., Kerr, D. S., and Orooji, A. (1983). A message-oriented implementation of a multi-backend database system (MBDS). *Database Machines,* Leilich and Missikoff (eds.), Springer-Verlag.

Demurjian, S. A., Hsiao, D. K., and Menon, M. J. (1986). A multi-backend database system for performance gains. capacity growth and hardware upgrade. *Proceedings of the Second International Conference on Data Engineering.*

He. X., Higashida, M., Hsiao, D. K.. Kerr. D. S., Orooji, A., Shi, Z. and Strawser, P. R. (1983). The implementation of a multi-backend database system (MBDS): part II - the first prototype MBDS and the software engineering experience. *Advanced Database Machine Architecture,* Hsiao (ed.), Prentice-Hall.

Kerr, D.S.. Orooji, A., Shi, Z. and Strawser, P. R. (1983). The implementation of a multi-backend database system (MBDS): part I - software engineering strategies and efforts towards a prototype MBDS. *Advanced Database Machine Architecture,* Hsiao (ed.), Prentice-Hall.

On portable database systems:

Silberman, B. (1986). Software portability: a case study of the multi-backend database system. Master's Thesis, Naval Postgraduate School, Monterey, California.

Wong, A. (1986). Towards highly portable database systems: issues and solutions. Master's Thesis, Naval Postgraduate School, Monterey, California.

On user interfaces for database systems:

Demurjian, S. A. and Hsiao, D. K. (1985). New directions in database systems research and development. *Proceedings of the International Symposium on New Directions in Computing.*

Demurjian, S. A. and Hsiao, D. K. (1986). A multi-lingual database system. Technical Report, NPS-52-86-011. Naval Postgraduate School. Monterey, California.

Wu, C. T. (1986). A new graphics user interface for accessing a database. *International Computer Graphics Conference.*

Wu. C. T. (1986). A simplicity requirement of a CAD/CAM information support system. *Workshop on Information System Support for Integrated Design and Manufacturing.*

On methodologies for database computers:

Demurjian, S. A. and Hsiao, D. K. (1985). Benchmarking database systems in multiple backend configurations. *IEEE Database Engineering Bulletin,* March.

Demurjian, S. A., Hsiao, D. K., Kerr, D. S., Menon M. J., Strawser, P. R., Tekampe, R. C., Watson, R. J. and Trimble, J. (1985). Performance evaluation of a database system in multiple backend configurations. *Proceedings of the 1985 International Workshop on Database Machines.*

Demurjian, S. A., Hsiao, D. K., Kerr, D. S., Tekampe, R. C. and Watson, R. J. (1985). Performance measurement methodologies for database systems. *Proceedings of the National ACM Conference.*

Demurjian, S. A., Hsiao, D. K. and Marshall, R. G. (1986). *Design analysis and performance evaluation methodologies for database computers.* To be published by Prentice-Hall.

Demurjian, S. A., Hsiao, D. K. and Strawser, P. R. (1986). Design analysis and performance evaluation methodologies for database computers. *Advances in Computers,* Yovits (ed.), Vol. 25, Academic Press.

Hsiao, D. K. and Menon, J. M. (1983). Design and analysis of join operations of database machines. "Advanced Database Machine Architecture," Hsiao (ed.), Prentice-Hall, 1983.

On Multi-Medium Database Systems:

Kropp, D. and Walch, G. (1981). A graph structured text field index based on word fragments. *Information Processing and Management,* Vol. 17, No. 6.

Lum, V., Dadam, P., Erbe, R., Guenauer, J., Pistor, P., Walch, G., Werner, H.-D. and Woodfill, J. (1984). Designing DBMS support for the time dimension. *Proceedings of the 1984 SIGMOD Conference.*

Lum, V., Dadam, P., Erbe, R., Guenauer, J., Pistor, P., Walch, G., Werner, H.-D. and Woodfill, J. (1985). Design of an integrated DBMS to support advanced applications. *Proceedings of the International Conference on Foundations of Data Organization.*

Scheck, H.-J. (1978). The reference string indexing method. *Information System Methodology,* Bracchi and Lockemann (eds.), Springer Verlag.

INITIAL DISTRIBUTION LIST

Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Dudley Knox Library Code 0142 Naval Postgraduate School Monterey, CA 93943-5100	2
Office of Research Administration Code 012 Naval Postgraduate School Monterey, CA 93943-5100	1
Chairman, Code 52V1 Computer Science Department Naval Postgraduate School Monterey, CA 93943-5100	40
David K. Hsiao Code 52Hq Computer Science Department Naval Postgraduate School Monterey, CA 93943-5100	200
Chief of Naval Research Arlington, VA 22217	1